Cryptanalysis of LASH

Scott Contini Krystian Matusiewicz Josef Pieprzyk
Ron Steinfeld Guo Jian Ling San Huaxiong Wang

October 2007

Table of contents

Motivation for LASH

- LASH is cryptographic hash function designed by Bentahar, Page, Saarinen, Silverman, and Smart.
- Published in NIST's second cryptographic hash workshop, 2006.
- Based upon provable design of Goldreich, Goldwasser, and Halevi (GGH) but modified in an attempt to make more practical and more secure.
- Aims: For output size x, should require $2^{\frac{x}{2}}$ work to find collisions and 2^x work to find preimages.
- LASH-x is proposed having output size of x bits, for $x = 160$, 256, 384 and 512.

Building upon GGH

- GGH is a provable design.
  - If algorithm exists to find collisions in GGH, then the algorithm can be used to find small vectors in a lattice.
  - Algorithm is effective for worst case lattice problems.
  - Since these lattice problems seem hard, we get a design for which it is hard to find collisions.
- LASH authors are not happy with GGH for two reasons:
  - Not efficient.
  - For GGH with x bit output, they claim it can be attacked in $2^{x/3}$ operations, or even $2^{x/4}$ for certain parameters.

GGH Construction

- Let H be an m by n matrix with entries in Z_q.
- Assume $m \log q < n < \frac{q}{2m^4}$ and $q = O(m^c)$ for const $c > 0$.
- Let message consist of bits $s_1, \ldots, s_n \in \{0,1\}^n$. Let s be vector consisting of these bits.
- Then hash is simply $h = Hs \bmod q$:

$$\begin{array}{cc} & n \text{ columns} \\ \begin{matrix} m \\ \text{rows} \end{matrix} & \overbrace{\begin{bmatrix} \cdots & H & \cdots \end{bmatrix}} \end{array} \cdot \begin{bmatrix} s_1 \\ \vdots \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} h_1 \\ \vdots \\ h_m \end{bmatrix}$$

Attacking GGH

- Despite provability, LASH authors claim that it can be attacked in $O(2^{x/3})$ when embedded in Merkle-Damgård iteration, where $x = m \log q$ is size of output.
- They sketch how to attack GGH when arithmetic is done over Z_{256}.
- Attack is time-memory tradeoff based upon Pollard iteration:
  - Choose messages such that the hashes are confined to some subspace S with size significantly smaller than 2^x.
  - After $\sqrt{|S|}$ iterations, a collision is expected (birthday paradox).
  - Find collision with Pollard rho method.

Attacking GGH: The Smaller Subspace

- One can force several bits of the hash output ($8m$ bits) to zero by choosing message blocks cleverly:
  - Note that Hs mod 2 is a linear system over $GF(2)$. With simple linear algebra, messages can be chosen so that the least significant output bits (m bits total) are zero.
  - A precomputed table is used to force further c bits to zero:
    - Table has 2^c entries and uses $m + c$ message bits (since table lookup entries must also have least significant bits set to zero).
    - Precomputation phase requires 2^c time.
  - So $m + c$ of $x = 8m$ bits are zero, yielding $|S| = 2^{7m-c}$.
  - Collision expected after $2^{\frac{1}{2}(7m-c)}$ iterations.
  - Balancing Pollard time $2^{\frac{1}{2}(7m-c)}$ with precomp time 2^c, optimal run time is $2^{7x/24}$.

Attacking GGH: Continued

- Authors claimed that attack can be done in $2^{x/4}$ (but only gave details of the $2^{7x/24}$ attack).
- If n (number of columns) is much greater than m^2 (square of number of rows), then least significant two bits can be forced to zero with linear algebra.
- This reduces S to size 2^{6m-c}.
- Setting $c = 2m = \frac{x}{4}$ gives optimal run time.

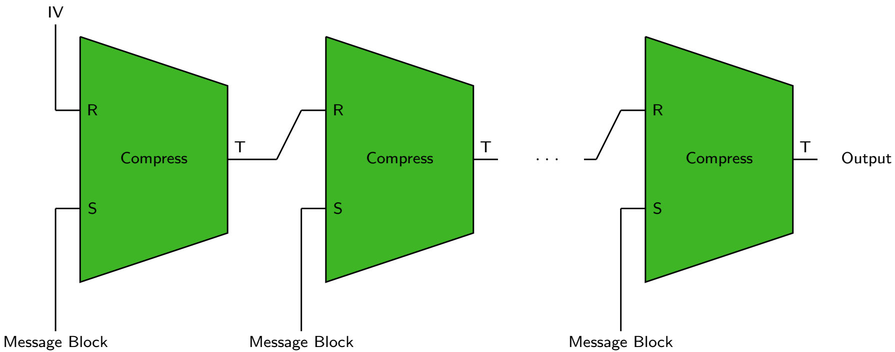
Subtleties of GGH Attack

- Attack requires large memory: 2^c precomp values must be stored. From a cost based analysis, attacks are inefficient.
- They only seem to have attacked GGH for invalid parameters:
  - Substituting $q = 256$ into $m \log q < n < \frac{q}{2m^4}$, we derive that $m < 2$. Yet they use $m \geq 40$.
  - The real GGH requires much larger matrix elements for security proof to hold.
- Nevertheless, this is the motivation for the LASH design:
  - They want to use $q = 256$ for efficiency.
  - GGH with $q = 256$ is not resistant to collision attacks faster than square root time.

From GGH to LASH

- Restrict to matrix elements to Z_{256}.
- Choose number of columns to be $n = 16m$.
- Add Miyaguchi-Preneel heuristic.
- Embed the design into Merkle-Damgård structure, including standard padding techniques and putting message length block at the end.
- Throw in Lucks' double-pipe heuristic.
  - They do so by adding final transform.
  - After processing final block (message length block), truncate off least significant half-bytes.
  - So length of hash is $x = 4m$.
- Choose an all zero initial vector (IV).

LASH in Merkle-Damgård Structure



LASH Final Transform

- Final Transform chops off least significant half-byte of every byte.

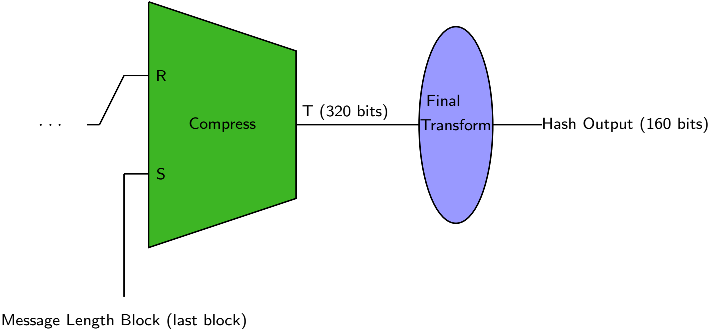


Diagram of LASH-160 Compression Function

$$\text{40 bytes}\left\{\begin{bmatrix} \\ R \\ \\ \end{bmatrix} \oplus \begin{bmatrix} \\ S \\ \\ \end{bmatrix} + \overbrace{\begin{bmatrix} & & \\ \cdots & H & \cdots \\ & & \end{bmatrix}}^{\text{640 columns}} \cdot \overbrace{\begin{bmatrix} r_0 \\ \vdots \\ r_{319} \\ \text{---} \\ s_0 \\ \vdots \\ s_{319} \end{bmatrix}}^{bitvec}\right.$$

PseudoCode for LASH Compression Function

Input: Vectors R, S (40 bytes each) and corresponding expanded vectors r and s (320 bits each).

For $j = 0$ **to** 39 **do**
  $T_j = R_j \oplus S_j$;
For $i = 0$ **to** 319 **do**
{
  If $(r_i == 1)$ **then**
    For $j = 0$ **to** 39 **do**
      $T_j = T_j + H_{i,j} \bmod 256$;
  If $(s_i == 1)$ **then**
    For $j = 0$ **to** 39 **do**
      $T_j = T_j + H_{i,j+320} \bmod 256$;
}
Return T;

A Fixed Point in LASH

- We demonstrate an attack against LASH very similar to the attack that LASH authors had on GGH.
- Note that an input of all zeros is a fixed point of the compression function.
  - If R and S are all zeros, then so is v, where v is vector consisting of bits of R and S.
  - Hence $R \oplus S + H \cdot v$ is all zeros.
- Because the IV is all zeros, we have complete control over the fixed point.
- We can send in as many all zero message blocks as we want, resulting in all zero intermediate outputs.

Exploiting the Fixed Point

- Consider messages of the following form:
  - Starting out with several zero message blocks (the exact number to be determined later).
  - Then one "random" message block.
- Irregardless of the number of zero blocks, the intermediate output of LASH applied to this message is a fixed value.
  - It is determined entirely by the one random block.
  - Number of zero blocks has no effect.
- The only reason why we do not have trivial collisions is because of message length block that is appended afterward.
- We will choose a message length that results in the end hash value having several bits equal to zero.

Precomputation

- Consider last application of compression function:
  - This is where the message length block is put in.
  - After this compression, final transform is applied.
- Let H_2 be the right hand side of H.
  - Corresponding to the bits of S that will hold message length.
- We consider 2^c different message lengths (Precomputation phase).
  - Only consider relatively small messages.
  - Hence most of bits of S are zero.
- Multiply H_2 by each encoded message length.
- Store resulting 40 byte vectors in a file.

Visualizing the Attack

- Use precomp table to determine message length ℓ so that lower most significant half-bytes of T are zero.

$$\overbrace{\begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}}^{R} \oplus \overbrace{\begin{bmatrix} \ell \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}^{S} + \left[\begin{array}{c|c} \quad H_1 \quad & \quad H_2 \quad \end{array}\right] \cdot \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ - \\ \ell \\ 0 \\ 0 \\ 0 \end{bmatrix} = \overbrace{\begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ 0|. \\ 0|. \end{bmatrix}}^{T}$$

Algebra of the Attack

- We have $T = R \oplus S + H_1 r + H_2 s$.
  - R (r) is fed in from previous iteration, so it known.
  - S (s) has zeros for all but the top bits.
- We can compute bottom bits of $R \oplus S + H_1 r$.
- Then use table lookup to determine a message length ℓ that is encoded to an s vector such that $R \oplus S + H_1 r + H_2 s$ has zeros in bottom most significant half-bytes of T.
  - If we have 2^c precomp, we can aim for bottom $c/4$ half-bytes to be zero.
  - Because of integer carries (mod256), there is a 50% chance that each half-byte will be 1 instead of 0.
  - Thus, we are only guaranteed $3c/4$ bits are zero.

Analysis

- Precomp takes 2^c time and 2^c memory.
- Each iteration sets $3c/4$ of $x = 8m$ bits to zero.
- Subspace is size $|S| = 2^{x-3c/4}$.
- Pollard iteration takes time $\sqrt{|S|} = 2^{x/2-3c/8}$.
- Balancing Pollard time with precomp, we solve $2^c = 2^{x/2-3c/8}$.
  - Solution is $c = (4/11)x$.
  - Running time is $2^{(4/11)x}$: slightly more than cube root.
- Similar idea works for finding preimages.
  - Preimages can be found in $2^{(4/7)x}$ time.

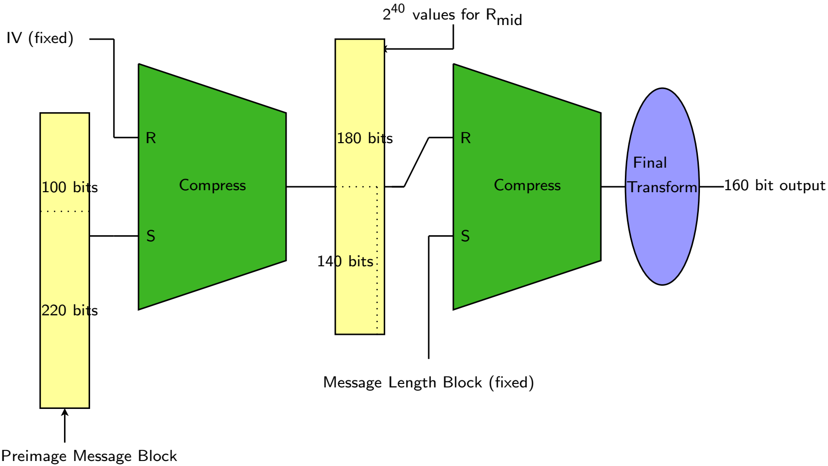
Additional Remarks

- Our paper has additional details (like how to deal with padding bit).
- We implemented this for LASH-160.
    - Using $c = 28$, an unoptimized implementation found messages colliding on the last 88 bits (11 bytes) of the hash.
        - Note that a naive Pollard iteration would take 2^{44} hashes to get the same result.
        - Each hash requires order $40 \times 320 > 2^{13}$ single precision computer operations.
        - So a naive search would have taken $> 2^{57}$ computer operations to get the same result.
        - We found our solution in a few days on a single Pentium.
- In theory LASH-160 can be broken in 2^{58} time/memory.

Can LASH be Patched?

- Our long message attacks can be prevented by changing the IV.
- So we consider attacks against LASH with arbitrary IV:
  - Preimages can be computed in $2^{7x/8}$ operations.
  - LASH compression function is trivially not a PRF when IV (or any subset of inputs) is replaced with a secret key.
    - PRF is needed in security proof for HMAC.

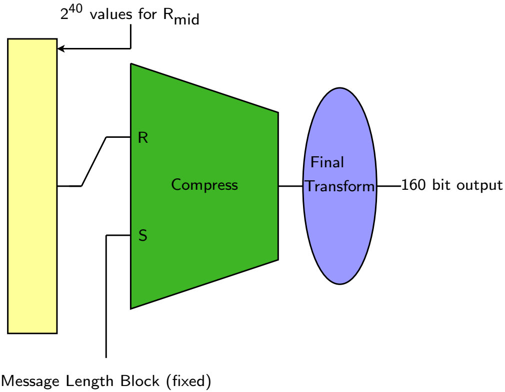
Visualizing Preimage Attack



Step 1 of Preimage Attack

- Given an output T of LASH-160, we compute 2^{40} values for R_{mid}, i.e. $FT\left[f(R_{mid}, S_{len})\right] = T$:
  - S_{len} corresponds to the encoded length block (fixed).
  - f represents LASH comrpession function.
  - FT is final transform.
- This is done with time/memory tradeoff, similar to long message attacks.
  - Use $c = 5x/7$, resulting in about 2^{114} time/memory.

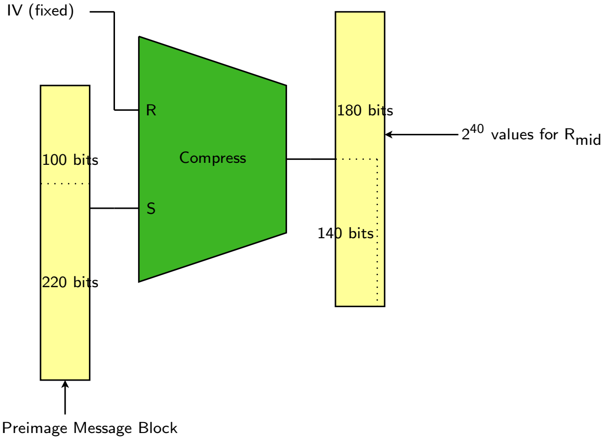
Visualizing Step 1 of Preimage Attack



Step 2 of Preimage Attack

- For each possibility of the first 100 bits of the message block and for each R_{mid} value:
  - Use "hybrid partial inversion algoirthm" to derive the remaining 220 bits of the message block M such that $f(IV, M)$ matches R_{mid} on the top 20 bytes and all least significant bits (180 bits total).
  - With probability 2^{-140}, it will match R_{mid} on the remaining 140 bits.
  - Since this iteration runs 2^{140} times, we expect one match, i.e. one preimage.
  - Running time is equivalent to 2^{140} calls to hybrid partial inversion algorithm.

Visualizing Step 2 of Preimage Attack



Hybrid Partial Inversion Algorithm Precomputations

- Since R is fixed (the IV), we can write $R \oplus S + Hv = T$ as $H' \cdot s =$ T' for some matrix H' and vector T'.
- We first prepare a precomp table involving the bottom 180 bits of s.
  - Loop through $2^{7x/8} = 2^{140}$ possibilities for the first 140 of these bits.
  - Do $GF(2)$ linear algebra to determine what last 40 bits should be so that adding corresponding selected columns of H' results in a vector y with zeros in all least significant bits.
  - Store bit vectors (180 bits of s) in hash table indexed by top 20 bytes of y.

Hybrid Partial Inversion Algorithm

- Given first 100 bits of s, we do $GF(2)$ linear algebra to determine next 40 bits of s so that adding corresponding selected columns of H' agrees with T' in all least significant bits.
  - Adding anything from hash table will preserve this agreement.
- Then use hash table to find remaining 180 bits of s so that the sum matches seven most significant bits of top 20 bytes of T'.
- Hence we match entire top 20 bytes and the least significant bits of remaining 20 bytes (180 bits).
- After 2^{140} tries, we expect to have a preimage!

LASH Compression Function is not a PRF

- Assume R (r) is some secret key.
- $f(R, S) = R \oplus S + H_1 r + H_2 s$.
- Set all bits of S to zero: $f(R, 0) = R + H_1 r$.
- Let S' have only first bit set and all others are zero.
- The $f(R, S') = f(R, 0) + W + H_{2,1}$ where
    - W is byte vector having only the most significant bit of the first byte set, and the remaining bits zero.
    - $H_{2,1}$ is the first column of H_2.
- Thus, regardless of secret R, one can distinguish from PRF in two queries.

Conclusion

- Design of LASH is motivated by GGH, but changed to avoid some questionable attacks and to improve practicality.
- Similar attacks to what LASH authors claimed against GGH also apply to LASH because of the naive choice of all zero IV.
- Even if the IV is changed, there are still attacks against LASH that make the design less than ideal.
- Is LASH preferable to GGH?
    - LASH is closer to practical but has no security reduction.
    - Designers traded provability for practicality.
    - Still, LASH is far too slow compared to designs like SHA-1, and is even slower than some provable designs.